Comparison of performance of data imputation methods in the context of their impact on the prediction efficiency of ranger Random Forest algorithm implementation

Ada Gąssowska, Mateusz Grzyb, Elżbieta Jowik

Abstract

It is very common for various datasets to contain incomplete observations. The analysis conducted by ignoring cases with missing data and considering only complete observations is perceived as naive, inefficient and exposed to bias, as missing values may also convey valuable information. To incorporate these sketchy observations, various methods of handling missing data may be considered, including both less and more sophisticated approaches. It turns out that the performance of machine learning classification models moderately depends on the type of approach that is applied to the imputation problem and the time of imputation, when using different techniques, is highly diverse. Presented article focuses on one basic technique, filling missing numeric values with median and categorical variables with mode, and five more sophisticated ones, which origin respectively from: mice, missRanger, VIM and softImpute R packages.

This paper introduces imputation techniques in the context of their impact on the prediction efficiency of ranger Random Forests algorithm implementation and the time of filling missing data.

The differences in the results of the classification algorithm for the imputation techniques in respect to measures under consideration are slight, nevertheless, the best outcome was obtained for the hotdeck method from the VIM package. As for the time of imputation, it highly depends on the technique complexity, so the basic method turns out to be the fastest one.

1 Introduction

Dealing with missing data is a substantial part of feature engineering as most of the machine learning algorithms do not accept incomplete datasets. However, despite the acknowledged importance of data imputation, in many practical cases it is not handled with proper caution. Basic median/mode imputation is often used, as well as deleting rows with missing data (if the number of missing values is low).

This article is concentrated on comparing these common simple methods with some of the more advanced algorithms (selected ones from VIM, missRanger, mice and softImpute R packages). For testing purposes, as the classification algorithm, we used the ranger Random Forests algorithm implementation, which is fast and particularly suited for high dimensional data.

The prediction effectiveness was assessed in relation to the area under the ROC curve (AUC), balanced accuracy (BACC) and Matthews correlation coefficient (MCC) measures. Time of each imputation was also measured, and taken into consideration.

2 Methodology

The ranger Random Forests algorithm implementation is particularly suited for high dimensional data. As it does not handle data with missing values, datasets must be imputed before classification.

We used six different imputation functions on each benchmarking dataset:

- **impute_basic()** using impute() function from imputeMissings package.
  It imputes missing numeric values with median and categorical variables with mode of other records.

- **impute_mice()** using mice() from mice

package.
The function uses Predictive Mean Matching to impute missing
values.

- **impute_missRanger()** using missRanger() function from missRanger package.
  The function uses the ranger package to do fast missing value imputation by chained random
  forest.

- **impute_VIM_hotdeck()** using hotdeck() function from VIM package.
  Each missing value is replaced with an observed response from a "similar" unit.

- **impute_VIM_knn()** using kNN() function from VIM package.
  It finds the k closest neighbors to the observation with missing data and then imputes them based on the the non-missing values from its neighbors.

- **impute_softImpute_mode()** using function softImpute() from softImpute package
  for numeric variables imputation and mode imputation for categorical variables imputation.

Function using imputeFAMD() from missMDA package was also created but it was unable to perform imputation on all of benchmarking datasets due to issues with convergence.
Imputation with use of Amelia package also proved impossible due to occurrence of highly correlated variables in some benchmarking datasets.

To automatize our work we created specialized functions: **split_and_impute()** and **train_and_test()**.

- **Split_and_impute()** divides data into train and test sets and imputes it with specified imputer. I takes four arguments:
  - id - id of benchmarking dataset to impute,
  - imputer - one of the described earlier imputation functions,
  - train_size - size of train set (values from 0 to 1, default value is 0.8),
  - save - whether to save imputed dataset (default value is TRUE)

  and returns:
  - train - imputed train dataset,
  - test - imputed test dataset,
  - time - time of imputation

- **Test_and_train()** performs crossvalidation on train set and makes the predictions for test set made with specified learner, trained on whole train set. Based on mentioned results it calculates AUC, BACC and MCC measures for both crossvalidation and test set testing stages. The function takes seven arguments:
  - train - imputed train dataset without missing values,
  - test - imputed test dataset without missing values,
  - learner - a classification learner which returns probabilities. In this case we will be using Ranger Random Forest learner,
  - target - name of target variable in the dataset,
  - postive - label of positive class (default value is '1'),
  - folds - number of folds used for crossvalidation (default value is 5),
  - title - title for crossvalidation metrics plot (default value is no title)

  and returns values needed for the comparison of methods of imputation:
  - cv_plot - plot of Receiver Operating Characteristics curve (ROC), AUC, BACC and MCC achieved during crossvalidation stage,
  - mean_auc - mean AUC achieved during crossvalidation stage,
  - mean_bacc - mean BACC achieved during crossvalidation stage,

- mean_mcc - mean MCC achieved during crossvalidation stage,
- test_auc - AUC achieved during test stage,
- test_bacc - BACC achieved during test stage,
- test_mmc - MCC achieved during test stage,
- learner - learner trained on whole train set.

After calling split_and_impute() and test_and_train() functions on each dataset with six different imputers we were able to compare performance of each tested imputation function in the context of their impact on the prediction efficiency of Ranger Random Forest algorithm.

3 Results

Each imputation function was performed on twelve benchmarking datasets with missing data, divided into test and train sets. We gathered the predictions for test sets from ranger Random Forest model fitted on train sets, and compared them using AUC, BACC and MCC (see Table1, Figure1, Figure2 and Figure3). We also measured the time of each imputation and compared it (Figure 4).

4 Conclusions

All of the imputers performances were relatively similar. The best mean AUC, mean BACC and mean MCC were all scored by VIM_hotdeck imputer. It also proved to be fast, with imputation times of a few seconds. Mice and VIM_knn imputer were only slightly worse, however Mice imputatation took a lot of time for large datasets (up to 9 hours). SoftImputer aquired the worst score regarding AUC measure and it the second worst regarding BACC and MCC. MissRanger's scores were slightly better, but not as good as any of the imputation from VIM package. Basic median/mode imputer was the quickest one, but its mean scores were the lowest (except of AUC where it scored second lowest one).

method	**mean AUC**	**mean BACC**	**mean MCC**
basic	0.897	0.821	0.657
mice	0.907	0.828	0.672
missRanger	0.898	0.827	0.669
softImpute	0.893	0.822	0.664
VIM_hotdeck	0.908	0.837	0.690
VIM_knn	0.904	0.832	0.678

Table 1: Mean AUC, BACC and MCC for each method

It seems that according to our reaserch the best imputer is hotdeck() function from VIM package. Its scores are the highest and its time performance relatively good. It is also worth to notice that basic imputer was not much worse than other functions, while its imputation time was very low even for large datasets.

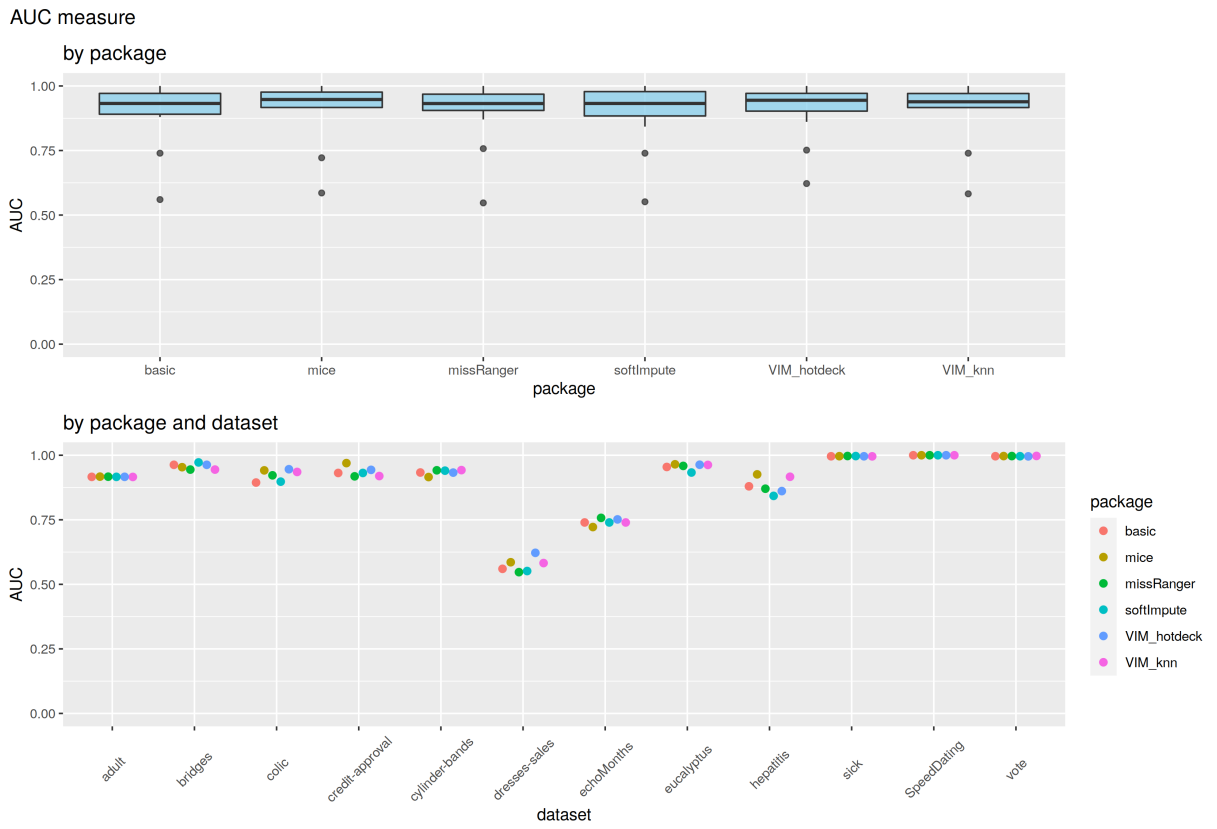

Figure 1: Area Under the Curve for different datasets and different imputations

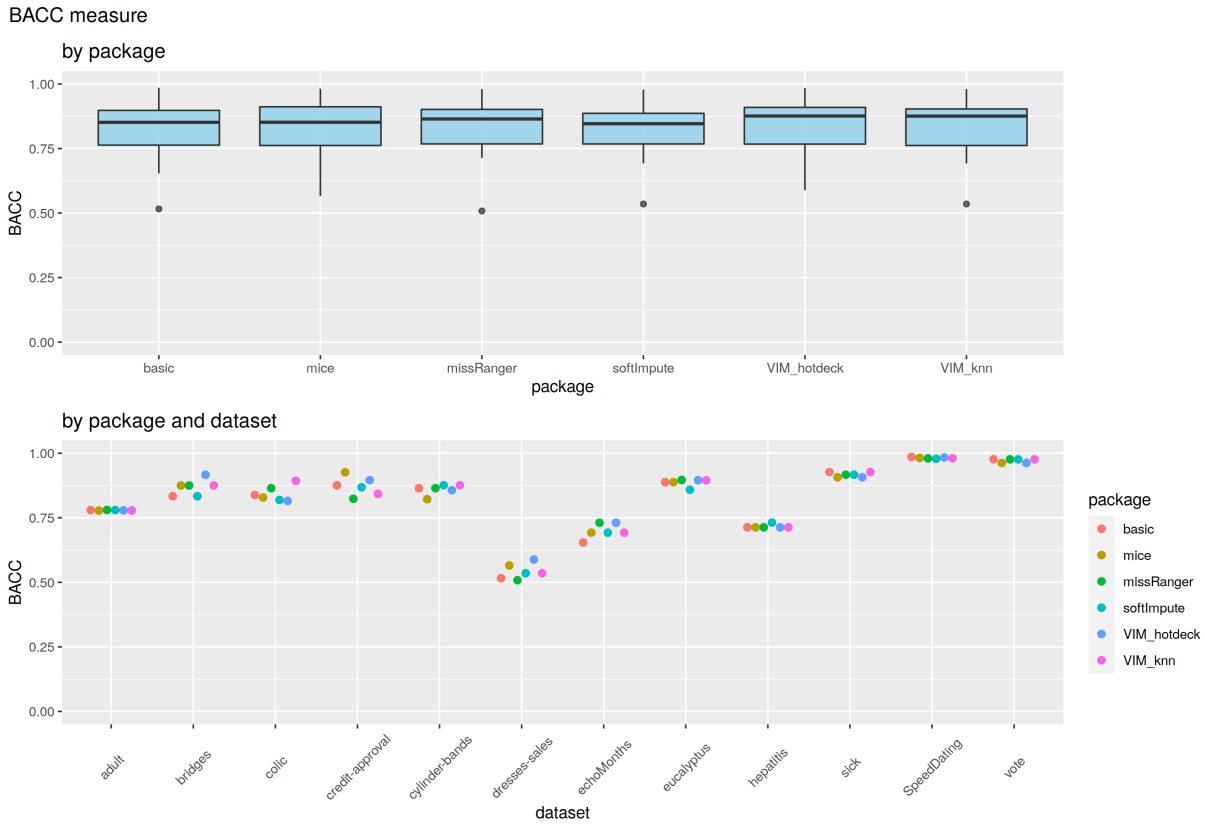

Figure 2: Balanced Accuracy for different datasets and different imputations

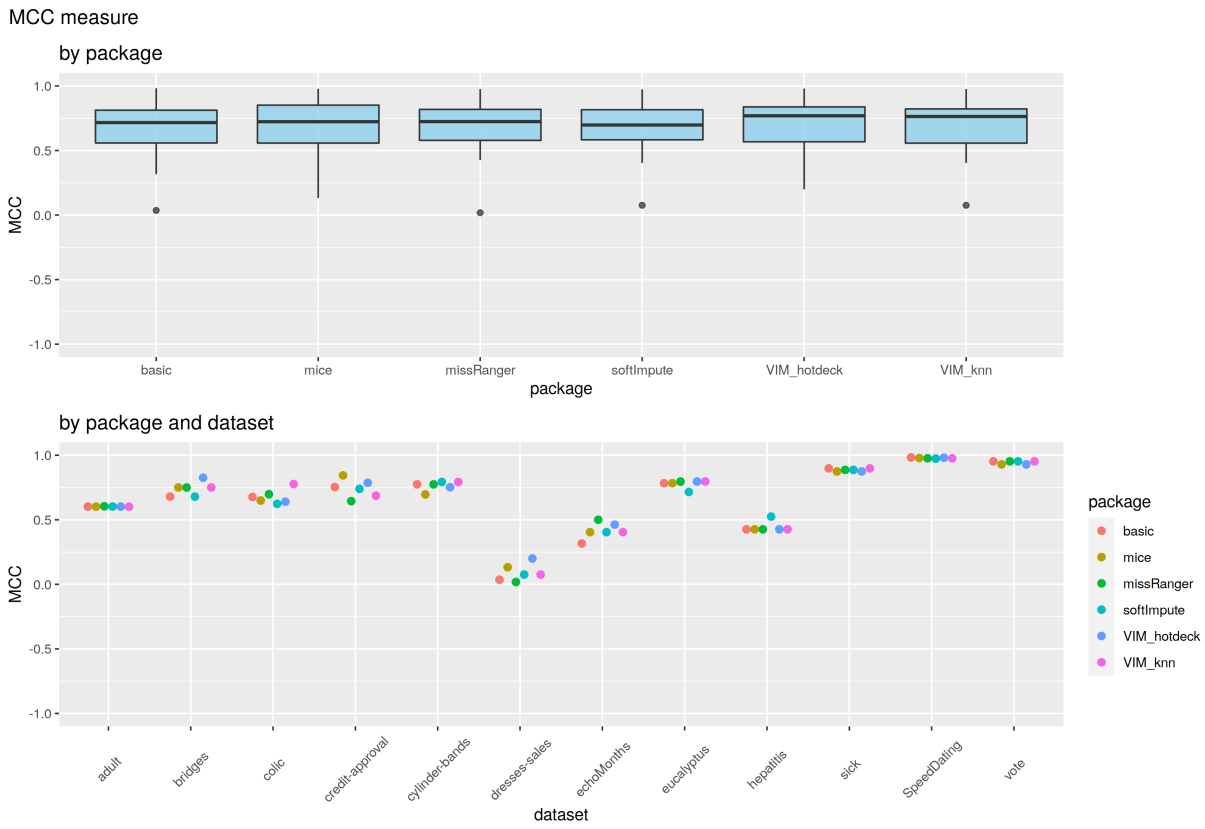

Figure 3: Matthews correlation coefficient for different datasets and different imputations

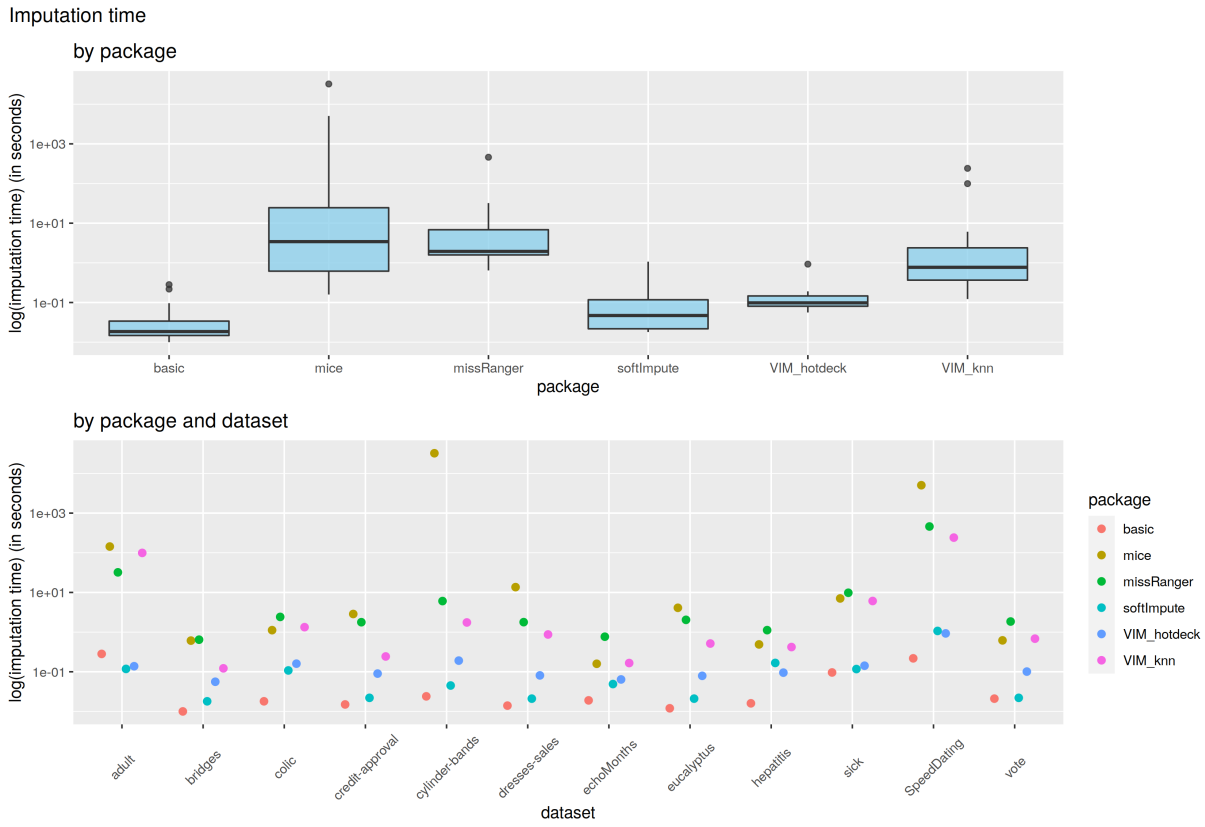

Figure 4: Logarithm of time of imputation for different datasets and different imputations